Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

**Высшая школа программной инженерии**

**КУРСОВАЯ РАБОТА**

**Автоматизация работы библиотеки, вариант №2**

по дисциплине «Системы управления базами данных»

Выполнил

студент гр.  3530904/90321            Афанасов Е. П.

Руководитель                                      Прокофьев О. В.

Санкт-Петербург

2022

**Оглавление**

[**Введение** 3](#_Toc100854688)

[**Описание реализации** 4](#_Toc100854689)

[**Результаты работы** 7](#_Toc100854690)

[**Вывод** 12](#_Toc100854691)

[**Источники** 13](#_Toc100854692)

[**Код программы** 14](#_Toc100854693)

# **Введение**

Цель курсовой работы – автоматизация работы библиотеки.

Библиотека, деятельность которой предлагается автоматизировать, является государственным предприятием и выдает литературу жителям Выборгского района (далее клиентам) на безвозмездной основе.

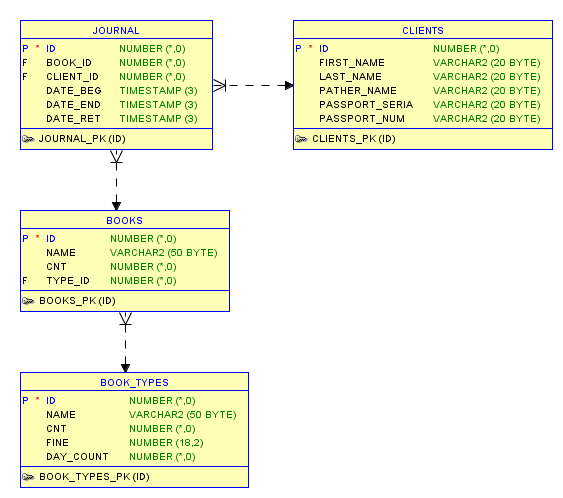
Клиенты характеризуются фамилией, именем и номером паспорта. Клиент не может иметь на руках более 10 книг.

Книги библиотеки характеризуются полным наименованием и разбиты на 3 категории: обычные, редкие и уникальные. Максимальные сроки удержания клиентом книги зависят от категории книги и составляют соответственно 60, 21 и 7 дней. При невозвращении клиентом книги в оговоренный срок, клиент обязан уплатить штраф из расчета 10, 50 и 300 рублей за день задержки так же в зависимости от категории книги. Экземпляров книг в библиотеке ограниченное количество и при выдаче и возврате их необходимо вести учет числа экземпляров каждой книги. Библиотекари при выдаче и приеме книг обязаны учитывать все операции в журнале.

При выдаче книги в журнал необходимо заносить дату выдачи (автоматически: текущая дата), клиента (библиотекарь выбирает из справочника), книгу (библиотекарь выбирает из справочника) и максимальную дату возврата (автоматически: текущая дата + количество дней, в зависимости от типа книги). При приеме книги устанавливается реальная дата возврата и, при необходимости, исчисляется штраф.

# **Описание реализации**

База данных для библиотеки была подготовлена в предыдущем семестре. Она имеет следующий вид:



Тогда же были реализованы:

- Функция, возвращающая количество книг на руках у читателей;

- Функция, высчитывающее сумму штрафа;

- Триггер, не дающий выдать книгу читателю при достижении лимита на количество книг на руках;

- Триггер на проверку паспортных данных нового читателя на случай совпадения с существующими читателями в базе;

- Триггер, который не позволяет удалить запись из журнала, если книга не возвращена;

- Триггер, проверяющий корректность даты возврата книги.

Для основной работы при автоматизации использовался следующий набор технологий:

Язык программирования: Python 3.8

Библиотека для работы с БД: psycopg 2.9.3

Библиотека для графического интерфейса: Tkinter

Библиотека для хэширования паролей: hashlib

Среда разработки: PyCharm 2021.2.3 (Community Edition)

Tkinter входит в набор стандартных библиотек Python и реализована на основе Tk — кроссплатформенной библиотеки базовых элементов графического интерфейса, распространяемая с открытыми исходными текстами, поэтому реализованный в рамках курсовой работы GUI будет работать на Windows, Linux, MacOS. Визуальные элементы заимствуются у текущей операционной системы, поэтому приложения, созданные с помощью Tkinter, выглядят так, как будто они принадлежат той платформе, на которой они работают.

Недостаток Tkinter – устаревший внешний вид интерфейсов, созданных на базе этой библиотеки.

Psycorg2 считается одним из самых стабильных, а потому и самых популярных модулей для работы с PostgreSQL. Его главная особенность заключается в полной реализации Python DB API 2.0 и безопасности потоков (несколько потоков могут использовать одно и то же соединение).

Структура программы:

- main.py – точка входа;

- Connection.py – выполняет подключение к БД и осуществляет проверку логина/пароля пользователя;

- MainWindow.py – основное окно системы, содержит таблицы с данными и принимает команды для работы с ними;

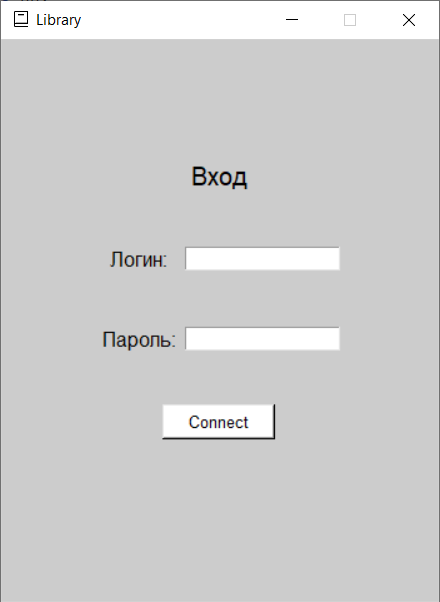
- MenuClients.py – реализация основных функций для работы с базой читателей – добавление, удаление.

- MenuJournal.py – реализация выдачи и возврата книг и сопутствующие им действия (изменение количества книг в библиотеке, уведомление о размере штрафа, вывод количества книг на руках).

- MenuBook.py – Добавление, удаление, изменение базы книг.

# **Результаты работы**

Первое окно при запуске программы – аутентификация пользователя:



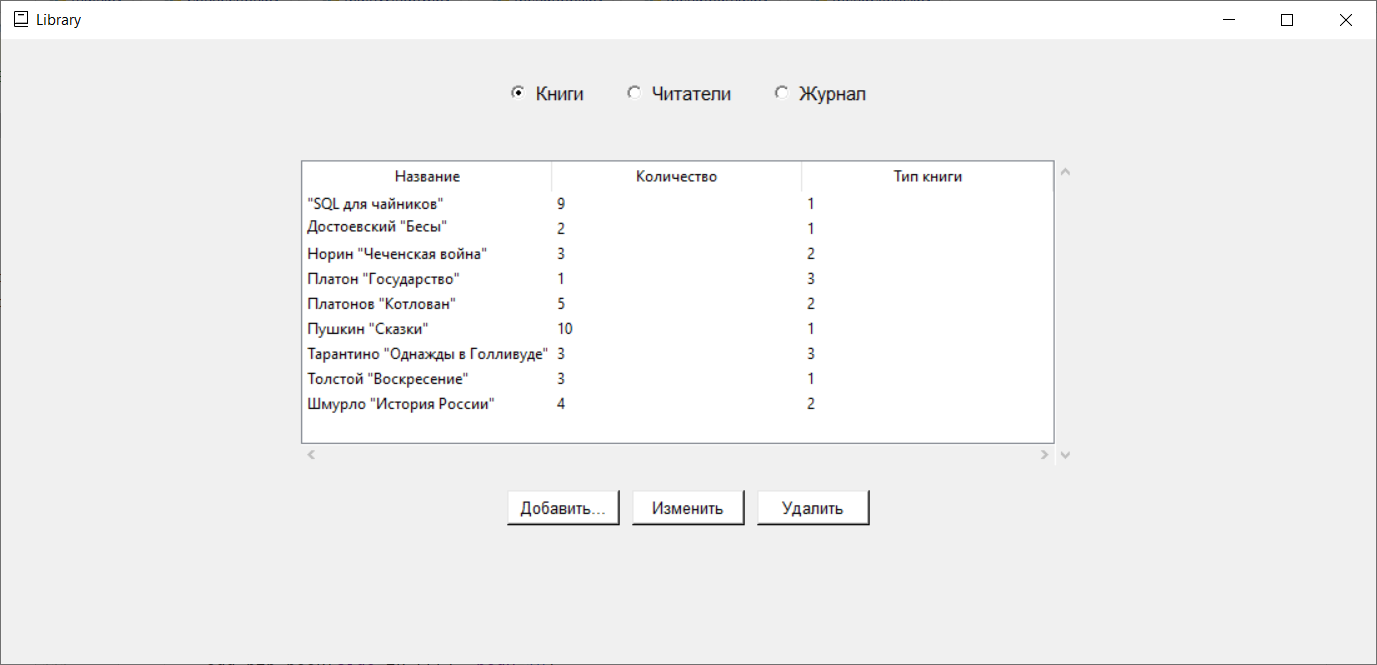
Неудачные попытки входа:

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Основное окно с переключателем таблиц и меню для действий:



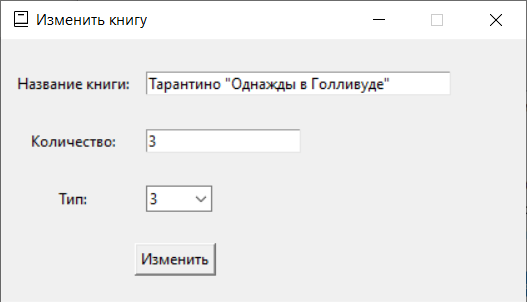
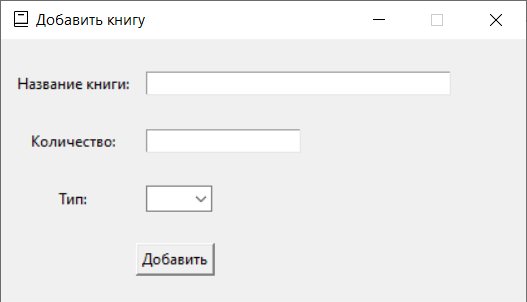
Изображение выглядит как стол

Автоматически созданное описание

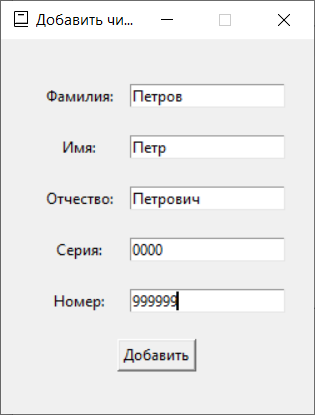
Изображение выглядит как стол

Автоматически созданное описание

Окна добавления и изменения книги:



Добавление читателя и срабатывание триггера:

Изображение выглядит как текст

Автоматически созданное описание

Предупреждение об удалении:

Изображение выглядит как текст

Автоматически созданное описание

Выдача книги – в списке только книги, которые есть в наличии, при выборе читателя выводится счетчик книг на руках:

Изображение выглядит как текст

Автоматически созданное описание

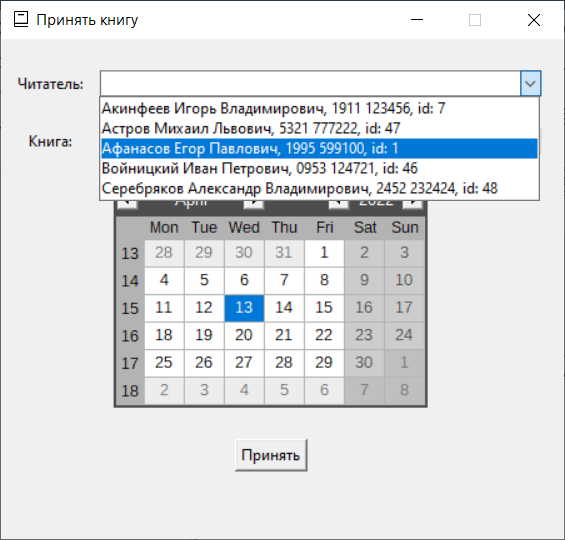
Изображение выглядит как текст

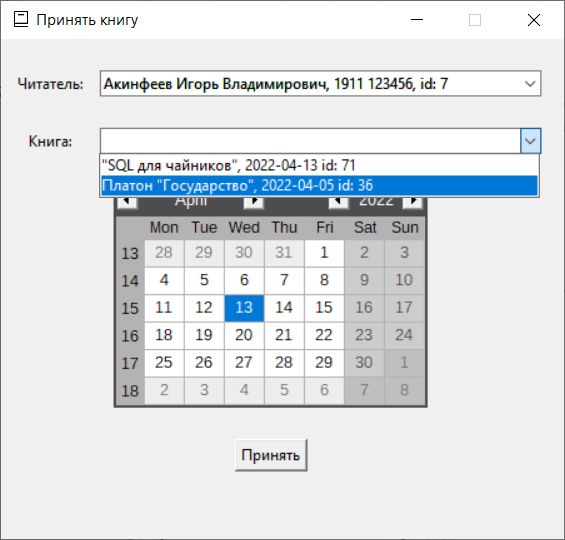
Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Прием книги (в списке только те читатели, у которых есть книги на руках):





Изображение выглядит как текст

Автоматически созданное описание

# **Вывод**

Была разработана система управления базой данных библиотеки. С помощью неё можно осуществлять ряд действий через интуитивно-понятный интерфейс: добавлять и удалять читателей, осуществлять прием и выдачу книг, актуализировать информацию о текущем каталоге.

В случае некорректных действий система их предотвратит. Например, библиотекарь не сможет удалить книги или читателей, связанных с незакрытыми записями в журнале. Невозможно добавить двух посетителей с одинаковыми паспортными данными, выдать читателю больше десяти книг на руки или установить дату возврата раньше даты выдачи.

Программа автоматически изменит количество книг в базе после изменений в журнале приема/выдачи. В случае несоблюдения сроков, установленных для возврата, библиотекарь будет уведомлен о штрафе для читателя и его размере.

Язык программирования Python 3.8 с использованием библиотек (psycopg2, tkinter) позволил осуществить проект в небольшие сроки с реализацией всех намеченных целей.

Tkinter не дает широких возможностей для настроек внешнего вида интерфейса, но этой библиотеки хватает для построения понятного и функционального приложения. Для библиотеки куда важнее предусмотреть возможные ошибки работников, нежели уникальный стиль.

# **Источники**

1. docs.python.org/3/library/tkinter.html
2. psycopg.org/docs/
3. pythonru.com/biblioteki/vvedenie-v-postgresql-s-python-psycopg2

# **Код программы**

|  |
| --- |
| **MainWindow.py** |
| import tkinter as tk import tkinter.ttk as ttk from MenuBook import MenuBook from MenuClients import MenuClients from MenuJournal import MenuJournal   class MainWindow:  def \_\_init\_\_(self, connection):  self.connection = connection  self.cursor = connection.cursor()  self.main = tk.Tk()  self.table\_frame = tk.Frame(self.main)  self.change\_frame = tk.Frame(self.main)  self.btn\_frame = tk.Frame(self.main)  self.menu\_frame = tk.Frame(self.main)  self.table\_state = tk.IntVar()  self.table\_state.set(0)   self.main\_window()   def main\_window(self):  self.main.geometry('1100x500')  self.main.title('Library')  self.main.iconbitmap('book\_education\_icon\_217331.ico')  self.change\_frame.pack(pady=30)   self.table\_frame.pack(pady=10)  self.menu\_frame.pack()   t1 = tk.Radiobutton(self.change\_frame, text='Книги', font=('Arial', 11), variable=self.table\_state, value=0,  command=self.change)  t1.pack(side=tk.LEFT)   t2 = tk.Radiobutton(self.change\_frame, text='Читатели', font=('Arial', 11), variable=self.table\_state, value=1,  command=self.change)  t2.pack(side=tk.LEFT, padx=25)   t3 = tk.Radiobutton(self.change\_frame, text='Журнал', font=('Arial', 11), variable=self.table\_state, value=2,  command=self.change)  t3.pack(side=tk.LEFT)   self.change()   self.main.mainloop()   def change(self):  for widget in self.table\_frame.winfo\_children():  widget.destroy()   for widget in self.menu\_frame.winfo\_children():  widget.destroy()   if self.table\_state.get() == 0:  self.show\_books()   elif self.table\_state.get() == 1:  self.show\_clients()   elif self.table\_state.get() == 2:  self.show\_journals()   def create\_table(self, sql\_req, columns, display\_columns, real\_names):  table = ttk.Treeview(self.table\_frame, show='headings', columns=columns, height=10)   scrl\_y = ttk.Scrollbar(self.table\_frame, orient="vertical", command=table.yview)  scrl\_y.pack(side=tk.RIGHT, fill='y')   scrl\_x = ttk.Scrollbar(self.table\_frame, orient="horizontal", command=table.xview)  scrl\_x.pack(side=tk.BOTTOM, fill='x')   table.configure(yscrollcommand=scrl\_y.set)  table.configure(xscrollcommand=scrl\_x.set)   for i in range(len(columns)):  table.heading(columns[i], text=real\_names[i])  table.column(columns[i], width=200)   self.cursor.execute(sql\_req)   for i in self.cursor.fetchall():  table.insert(parent='', index='end', text='',  values=i)   table['displaycolumns'] = display\_columns   return table   def check\_update(self, menu\_class):  if menu\_class.changed:  self.change()  elif menu\_class.state != self.table\_state.get():  return  else:  self.main.after(1000, lambda: self.check\_update(menu\_class))   def show\_books(self):  columns = ("id", "Name", "count", "Type")  real\_names = ("id", "Название", "Количество", "Тип книги")  req = """ select \* from books order by books."NAME" """   table = self.create\_table(req, columns, columns[1:], real\_names)  table.pack()   manage\_books = MenuBook(self.connection, table, self.main)   self.check\_update(manage\_books)   add\_btn = tk.Button(self.menu\_frame, text='Добавить...', font=('Arial', 10), bg='white',  command=manage\_books.add\_book, width=10)  add\_btn.pack(side=tk.LEFT, pady=10)   change\_btn = tk.Button(self.menu\_frame, text='Изменить', font=('Arial', 10), bg='white',  command=manage\_books.change\_book, width=10)  change\_btn.pack(side=tk.LEFT, padx=10)   delete\_btn = tk.Button(self.menu\_frame, text='Удалить', font=('Arial', 10), bg='white',  command=manage\_books.delete\_book, width=10)  delete\_btn.pack(side=tk.LEFT)   def show\_clients(self):  columns = ("id", "First\_Name", "Last\_Name", "Pat\_Name", "Passport\_Seria", "Passport\_Num")  real\_names = ("id", "Имя", "Фамилия", "Отчество", "Паспорт серия", "Паспорт номер")  req = """select \* from clients order by "LAST\_NAME" """   table = self.create\_table(req, columns, columns[1:], real\_names)  table.pack()   manage\_clients = MenuClients(self.connection, table, self.main)   self.check\_update(manage\_clients)   add\_client\_btn = tk.Button(self.menu\_frame, text='Добавить читателя...', font=('Arial', 10), bg='white',  command=manage\_clients.add\_client)  add\_client\_btn.pack(side=tk.LEFT, pady=10)   delete\_btn = tk.Button(self.menu\_frame, text='Удалить читателя', font=('Arial', 10), bg='white',  command=manage\_clients.delete\_client)  delete\_btn.pack(side=tk.LEFT, padx=10)   def show\_journals(self):  columns = ("id", "First\_name", "Last\_name", "Patr\_name", "Book\_name", "Date\_beg", "Date\_end", "Date\_ret")   real\_names = (  "id", "Имя", "Фамилия", "Отчество", "Название книги", "Дата выдачи",  "Вернуть до", "Дата возвращения")   req = """select journal."ID", clients."FIRST\_NAME", clients."LAST\_NAME", clients."PATHER\_NAME", books."NAME",  CAST(journal."DATE\_BEG" as date), CAST(journal."DATE\_END" as date), CAST(journal."DATE\_RET" as date)  from ((clients join journal on clients."ID" = journal."CLIENT\_ID") join  books on books."ID"=journal."BOOK\_ID") order by journal."DATE\_RET" DESC, journal."DATE\_BEG" DESC"""   table = self.create\_table(req, columns, columns[1:], real\_names)  table.pack()   manage\_journal = MenuJournal(self.connection, table, self.main)   self.check\_update(manage\_journal)   add\_client\_btn = tk.Button(self.menu\_frame, text='Выдать книгу', font=('Arial', 10), bg='white',  command=manage\_journal.add\_record)  add\_client\_btn.pack(side=tk.LEFT, pady=10)   return\_btn = tk.Button(self.menu\_frame, text='Принять книгу', font=('Arial', 10), bg='white',  command=manage\_journal.return\_book)  return\_btn.pack(side=tk.LEFT, pady=10, padx=10) |
| Connection.py |
| import psycopg2 import hashlib import tkinter.messagebox as box import tkinter as tk from MainWindow import MainWindow  *# 2 пользователя в БД (логин - пароль): # Egor - test12345 # usertest - puser* class Connection:  def \_\_init\_\_(self):  self.enter\_window = tk.Tk()   def hello\_window(self):  try:  connection = psycopg2.connect(user="postgres",  password="puser",  host="127.0.0.1",  port="5432",  database="libraryDB")  except psycopg2.OperationalError:  box.showerror("Ошибка", "Проблемы с подключением к базе данных")   self.enter\_window.title('Library')  self.enter\_window.iconbitmap('book\_education\_icon\_217331.ico')  self.enter\_window['bg'] = '#ccc'  self.enter\_window.geometry('350x450')  self.enter\_window.resizable(width=False, height=False)   frame = tk.Frame(self.enter\_window, bg='#ccc')  frame.pack(pady=75)   title = tk.Label(frame, text='Вход', bg='#ccc', font=('Arial', 15))  title.grid(row=0, column=0, columnspan=2, pady=20)   label\_login = tk.Label(frame, text='Логин: ', bg='#ccc', font=('Arial', 12))  label\_login.grid(row=1, column=0, pady=20)  login\_input = tk.Entry(frame, bg='white')  login\_input.grid(row=1, column=1, pady=20)   label\_pass = tk.Label(frame, text='Пароль: ', bg='#ccc', font=('Arial', 12))  label\_pass.grid(row=2, column=0, pady=20)  pass\_field = tk.Entry(frame, bg='white', show='\*')  pass\_field.grid(row=2, column=1, pady=20)   btn = tk.Button(frame, text='Connect', font=('Arial', 10), bg='white',  command=lambda: self.connect(login\_input.get(), pass\_field.get(), connection),  width=10)  btn.grid(row=3, column=0, columnspan=2, pady=20)   self.enter\_window.mainloop()   def connect(self, user, password, connection):  try:  curs = connection.cursor()  password = hashlib.md5(password.encode()).hexdigest()  curs.execute(f"select \"password\" from users where \"login\"='{user}'")  db\_password = curs.fetchone()[0]   if password == db\_password:  self.enter\_window.destroy()  MainWindow(connection)  else:  box.showerror("Ошибка в пароле", "Неверный пароль")  except TypeError:  box.showerror("Ошибка в логине", "Такой пользователь отсутствует")  except psycopg2.InternalError:  box.showerror("Ошибка", "Какая-то ошибка, обратитесь к Егору") |
| MenuBook.py |
| import tkinter as tk import tkinter.ttk as ttk import tkinter.messagebox as box  import psycopg2   class MenuBook:  def \_\_init\_\_(self, connection, table, main):  self.main = main  self.connection = connection  self.cursor = self.connection.cursor()  self.table = table  self.changed = False  self.state = 0   def insert\_book(self, name, count, types):  try:  self.cursor.execute(f"insert into books values (DEFAULT, '{name}', {count}, {types})")  self.connection.commit()  self.changed = True  except:  box.showerror("Ошибочка", "Incorrect data")  self.connection.rollback()  finally:  self.book\_window.destroy()   def update\_book(self, id, name, count, types):   try:  self.cursor.execute(  f"update books set \"NAME\"='{name}', \"CNT\"={count}, \"TYPE\_ID\"={types} where \"ID\"={id}")  self.connection.commit()  self.changed = True  except:  box.showerror("Ошибочка", "Incorrect data")  self.connection.rollback()  finally:  self.book\_window.destroy()   def add\_book(self):  self.book\_window = tk.Toplevel(self.main)  self.book\_window.grab\_set()  self.book\_window.geometry('420x210')  self.book\_window.title('Добавить книгу')  self.book\_window.iconbitmap('book\_education\_icon\_217331.ico')  self.book\_window.resizable(width=False, height=False)   name\_book\_label = tk.Label(self.book\_window, text='Название книги:')  name\_book\_label.grid(row=0, column=0, pady=25, padx=10)  name\_book = tk.Entry(self.book\_window, bg='white', width=40)  name\_book.grid(row=0, column=1, columnspan=3, sticky='W')   count\_books\_label = tk.Label(self.book\_window, text='Количество:')  count\_books\_label.grid(row=1, column=0, pady=0, padx=10)  count\_books = tk.Entry(self.book\_window, bg='white')  count\_books.grid(row=1, column=1, sticky='W')   type\_books\_label = tk.Label(self.book\_window, text='Тип:')  type\_books\_label.grid(row=2, column=0, pady=25, padx=10)  type\_books = ttk.Combobox(self.book\_window, values=[1, 2, 3], width=5, state='readonly')  type\_books.grid(row=2, column=1, sticky='W')   tk.Button(self.book\_window, text='Добавить',  command=lambda: self.insert\_book(name\_book.get(), count\_books.get(),  type\_books.get())).grid(row=3,  columnspan=2)   def change\_book(self):  info = self.table.item(self.table.selection())['values']  if len(info) == 0:  return   self.book\_window = tk.Toplevel(self.main)  self.book\_window.grab\_set()  self.book\_window.geometry('420x210')  self.book\_window.title('Изменить книгу')  self.book\_window.iconbitmap('book\_education\_icon\_217331.ico')  self.book\_window.resizable(width=False, height=False)   name\_book\_label = tk.Label(self.book\_window, text='Название книги:')  name\_book\_label.grid(row=0, column=0, pady=25, padx=10)  name\_book = tk.Entry(self.book\_window, bg='white', width=40)  name\_book.insert(0, f'{info[1]}')  name\_book.grid(row=0, column=1, columnspan=3, sticky='W')   count\_books\_label = tk.Label(self.book\_window, text='Количество:')  count\_books\_label.grid(row=1, column=0, pady=0, padx=10)  count\_books = tk.Entry(self.book\_window, bg='white')  count\_books.insert(0, f'{info[2]}')  count\_books.grid(row=1, column=1, sticky='W')   type\_books\_label = tk.Label(self.book\_window, text='Тип:')  type\_books\_label.grid(row=2, column=0, pady=25, padx=10)  type\_books = ttk.Combobox(self.book\_window, values=[1, 2, 3], width=5, state='readonly')  type\_books.set(info[3])  type\_books.grid(row=2, column=1, sticky='W')   tk.Button(self.book\_window, text='Изменить',  command=lambda: self.update\_book(info[0], name\_book.get(), count\_books.get(),  type\_books.get())).grid(row=3,  columnspan=2)   def delete\_book(self):  info = self.table.item(self.table.selection())['values']  if len(info) == 0:  return   if box.askyesno("Удаление", f"Удалить книгу {info[1]} и все записи в журнале, связанные с ней?"):  try:  self.cursor.execute(f"delete from journal where \"BOOK\_ID\"={info[0]}")  self.cursor.execute(f"delete from books where \"ID\"={info[0]}")  self.connection.commit()  self.changed = True  except psycopg2.InternalError:  box.showerror("Журнал", "Экземпляры этой книги находятся у читателей")  self.connection.rollback()  except:  box.showerror("Ошибочка", "Что-то пошло не так")  self.connection.rollback() |
| MenuClients.py |
| import tkinter as tk import tkinter.messagebox as box  import psycopg2   class MenuClients:  def \_\_init\_\_(self, connection, table, main):  self.connection = connection  self.cursor = self.connection.cursor()  self.table = table  self.main = main  self.changed = False  self.state = 1   def insert\_client(self, name, surname, patronymic, series, num):  if len(series) != 4 or len(num) != 6:  box.showerror("Error", "Некорректные пасспортные данные!")  return   try:  self.cursor.execute(  f"insert into clients values (DEFAULT, '{name}', '{surname}', '{patronymic}', '{series}', '{num}')")  self.connection.commit()  self.changed = True  except psycopg2.InternalError:  box.showerror("Ошибка", "Читатель с такими паспортными данными уже существует")  self.connection.rollback()  except ZeroDivisionError:  box.showerror("Ошибочка", "Incorrect data")  self.connection.rollback()  finally:  self.clients\_window.destroy()   def add\_client(self):  self.clients\_window = tk.Toplevel(self.main)  self.clients\_window.grab\_set()  self.clients\_window.geometry('250x300')  self.clients\_window.title('Добавить')  self.clients\_window.iconbitmap('book\_education\_icon\_217331.ico')  self.clients\_window.resizable(width=False, height=False)   entry\_frame = tk.Frame(self.clients\_window)  entry\_frame.pack(pady=15)   surname\_label = tk.Label(entry\_frame, text='Фамилия:')  surname\_label.grid(row=0, column=0, pady=20, padx=10)  surname = tk.Entry(entry\_frame, bg='white')  surname.grid(row=0, column=1)   name\_label = tk.Label(entry\_frame, text='Имя:')  name\_label.grid(row=1, column=0, pady=0, padx=10)  name = tk.Entry(entry\_frame, bg='white')  name.grid(row=1, column=1)   patronymic\_label = tk.Label(entry\_frame, text='Отчество:')  patronymic\_label.grid(row=2, column=0, pady=20, padx=10)  patronymic = tk.Entry(entry\_frame, bg='white')  patronymic.grid(row=2, column=1)   series\_label = tk.Label(entry\_frame, text='Серия:')  series\_label.grid(row=3, column=0, pady=0, padx=10)  series = tk.Entry(entry\_frame, bg='white')  series.grid(row=3, column=1)   num\_label = tk.Label(entry\_frame, text='Номер:')  num\_label.grid(row=4, column=0, pady=20, padx=10)  num = tk.Entry(entry\_frame, bg='white')  num.grid(row=4, column=1)   tk.Button(entry\_frame, text='Добавить',  command=lambda: self.insert\_client(name.get(), surname.get(),  patronymic.get(), series.get(), num.get())).grid(row=5,  columnspan=2)   def delete\_client(self):  info = self.table.item(self.table.selection())['values']  if len(info) == 0:  return   if box.askyesno("Удаление", f"Удалить читателя {info[1]} {info[2]} {info[3]} и все записи в журнале, связанные с ним?"):  try:  self.cursor.execute(f"delete from journal where \"CLIENT\_ID\"={info[0]}")  self.cursor.execute(f"delete from clients where \"ID\"={info[0]}")  self.connection.commit()  self.changed = True  except psycopg2.InternalError:  box.showerror("Журнал", f"Читатель {info[1]} {info[2]} {info[3]} не вернул книги")  self.connection.rollback()  except:  box.showerror("Ошибочка", "Что-то пошло не так")  self.connection.rollback() |
| MenuJournal.py |
| import tkinter as tk import tkinter.ttk as ttk import tkinter.messagebox as box  import psycopg2 import tkcalendar import datetime   class MenuJournal:  def \_\_init\_\_(self, connection, table, main):  self.connection = connection  self.cursor = self.connection.cursor()  self.table = table  self.number\_books = int()  self.main = main  self.changed = False  self.state = 2   def insert\_record(self, book\_id, client\_id):  try:  if self.number\_books >= 10:  box.showerror("Превышен лимит", "На руках у читателя уже 10 книг")  return   book\_id = book\_id.split()[0][:-1]  client\_id = client\_id.split()[0][:-1]   self.cursor.execute(f"select \"TYPE\_ID\" from books where \"ID\"={book\_id}")  type\_id = self.cursor.fetchone()[0]   self.cursor.execute(f"select \"DAY\_COUNT\" from book\_types where \"ID\"={type\_id}")  days = self.cursor.fetchone()[0]   date\_end = datetime.datetime.now().date() + datetime.timedelta(days=days)   self.cursor.execute(  f"insert into journal values (DEFAULT, '{book\_id}', {client\_id}, '{datetime.datetime.now()}', '{date\_end}', DEFAULT)")   self.cursor.execute(f"update books set \"CNT\"=\"CNT\"-1 where \"ID\"={book\_id}")   self.connection.commit()  self.changed = True  except ZeroDivisionError:  box.showerror("Ошибочка", "Incorrect data")  self.connection.rollback()  finally:  self.journal\_window.destroy()   def add\_record(self):  self.journal\_window = tk.Toplevel(self.main)  self.journal\_window.geometry('580x200')  self.journal\_window.title('Выдать книгу')  self.journal\_window.iconbitmap('book\_education\_icon\_217331.ico')  self.journal\_window.resizable(width=False, height=False)  self.journal\_window.grab\_set()   self.number\_books = "--"   self.cursor.execute("select \* from books where \"CNT\">0 order by \"ID\"")  books = self.cursor.fetchall()   self.cursor.execute("select \* from clients order by \"ID\"")  clients = self.cursor.fetchall()   *# print(books)* values\_book = []   for record in books:  values\_book.append(f"{record[0]}: {record[1]}")   *# print(clients)* values\_client = []  for record in clients:  values\_client.append(f"{record[0]}: {record[1]} {record[2]} {record[3]} {record[4]} {record[5]}")   books\_label = tk.Label(self.journal\_window, text='Книга:')  books\_label.grid(row=0, column=0, pady=25, padx=10)  books\_list = ttk.Combobox(self.journal\_window, values=values\_book, state='readonly', width=55)  books\_list.grid(row=0, column=1)   clients\_label = tk.Label(self.journal\_window, text='Читатель:')  clients\_label.grid(row=1, column=0, padx=10)  clients\_list = ttk.Combobox(self.journal\_window, values=values\_client, state='readonly', width=55)  clients\_list.grid(row=1, column=1)  clients\_list.bind("<<ComboboxSelected>>", self.count\_books)  how\_many\_books = tk.Label(self.journal\_window, text=f"Книг на руках: {self.number\_books}")  how\_many\_books.grid(row=1, column=2, padx=25, pady=25)   tk.Button(self.journal\_window, text='Добавить',  command=lambda: self.insert\_record(books\_list.get(),  clients\_list.get())).grid(row=3, columnspan=2)   def count\_books(self, event):  id = int(event.widget.get().split()[0][:-1])  self.cursor.execute(f"select \"count\" from how\_many\_books where \"ID\"={id}")  self.number\_books = self.cursor.fetchone()[0]  self.journal\_window.children['!label3']['text'] = f"Книг на руках: {self.number\_books}"   def return\_book(self):  def create\_list(event):  id = int(event.widget.get().split()[-1])  self.cursor.execute(f"select \"BOOK\_ID\", books.\"NAME\", \"DATE\_BEG\", journal.\"ID\" " +  f"from journal join books on \"BOOK\_ID\"=books.\"ID\" " +  f"where \"CLIENT\_ID\"={id} and \"DATE\_RET\" is null order by books.\"NAME\"")   records\_list = [k for k in self.cursor.fetchall()]  *# print(records\_list)* for k in range(len(records\_list)):  records\_list[k] = f"{records\_list[k][1]}, {records\_list[k][2].date()} id: {records\_list[k][-1]}"   record['state'] = 'readonly'  record['value'] = records\_list   self.journal\_window = tk.Toplevel(self.main)  self.journal\_window.grab\_set()  self.journal\_window.geometry('450x400')  self.journal\_window.title('Принять книгу')  self.journal\_window.iconbitmap('book\_education\_icon\_217331.ico')  self.journal\_window.resizable(width=False, height=False)   self.cursor.execute(f"select \* from how\_many\_books where \"count\">0 order by \"LAST\_NAME\"")   clients = self.cursor.fetchall()  *# print(clients)* for i in range(len(clients)):  clients[  i] = f"{clients[i][2]} {clients[i][1]} {clients[i][3]}, {clients[i][-3]} {clients[i][-2]}, id: {clients[i][0]}"   clients\_label = tk.Label(self.journal\_window, text='Читатель:')  clients\_label.grid(row=0, column=0, pady=25, padx=10)  clients\_list = ttk.Combobox(self.journal\_window, values=clients, state='readonly', width=55)  clients\_list.grid(row=0, column=1)  clients\_list.bind("<<ComboboxSelected>>", create\_list)   record\_label = tk.Label(self.journal\_window, text='Книга:')  record\_label.grid(row=1, column=0, padx=10)  record = ttk.Combobox(self.journal\_window, state='disabled', width=55)  record.grid(row=1, column=1)   calendar\_selector = tkcalendar.Calendar(self.journal\_window, selectmode='day')  calendar\_selector.grid(row=2, column=0, columnspan=2, pady=25, padx=0)   tk.Button(self.journal\_window, text='Принять',  command=lambda: self.update\_record(record.get().split()[-1], calendar\_selector.get\_date())).grid(  row=3, columnspan=2)   def update\_record(self, record\_id, date):  try:  self.cursor.execute(f"update journal set \"DATE\_RET\"='{date}' where \"ID\"={record\_id}")   self.cursor.execute(  f"update books set \"CNT\"=\"CNT\"+1 where \"ID\" in (select distinct \"BOOK\_ID\" from journal where journal.\"ID\"={record\_id})")   self.connection.commit()  self.changed = True   self.cursor.execute(  f"select sum(book\_types.\"FINE\" \* (CAST(journal.\"DATE\_RET\" as date) - CAST (journal.\"DATE\_END\" as date))) from journal LEFT JOIN books on journal.\"BOOK\_ID\"=books.\"ID\" LEFT JOIN book\_types on books.\"TYPE\_ID\"=book\_types.\"ID\" where journal.\"ID\"={record\_id}")  fine = self.cursor.fetchone()[0]   if fine > 0:  box.showwarning("ШТРАФ", f"Сумма штрафа: {fine} рублей")   except psycopg2.InternalError:  box.showerror("Ошибка даты", "Книга возвращена раньше даты выдачи")  self.connection.rollback()  except:  box.showerror("Ошибка", "Какая-то неизвестная доселе ошибка")  self.connection.rollback()  finally:  self.journal\_window.destroy() |
| main.py |
| from Connection import Connection  connect = Connection() connect.hello\_window() |